The homework is solved by writing:

```
obtain ⟨c, d, c_on_α, d_on_α, c_on_β, d_on_β, c_ne_d⟩ := Step4
```

We started new work with Step5:

```
have Step5 : Pt_on_Circ c α ↔ dist a c = dist a b :=
    Pt_on_Circ_iff a b c α a_center_of_α b_on_α
```

The iff statement is actually a bundle of two statements, `Pt_on_Circ c α → dist a c = dist a b` and `dist a c = dist a b → Pt_on_Circ c α`. To get the first one, we add a `().1` and that will give the first direction. In other words:

```
have Step5 : Pt_on_Circ c α → dist a c = dist a b :=
    (Pt_on_Circ_iff a b c α a_center_of_α b_on_α).1
```

So now Step5 has become:

```
have Step5 : dist a c = dist a b :=
    (Pt_on_Circ_iff a b c α a_center_of_α b_on_α).1 c_on_α
```

We made Step6:

```
have Step6 : dist b a = dist b c :=
    (Pt_on_Circ_iff b c a β b_center_of_β c_on_β).1 a_on_β
```

and observed that there are difficulties with ordering the points in distance, which shouldn't matter. So we're missing an axiom!

```
    dist_symm : ∀ (a b : Pt), dist a b = dist b a
```

We are now in the end game, prepared to attack the actual goal

```
⊢ ∃ c, IsEquilateralTriangle a b c
```

We want to specify what `Pt` to use to make progress. The syntax for that is... `use`
The goal, after `use c` has become:

```
⊢ IsEquilateralTriangle a b c
```

and I don't remember how `IsEquilateralTriangle` is defined. Let's unfold the definition. The syntax for that is: `unfold`. After `unfold IsEquilateralTriangle`, the goal state is to prove two things:

```
⊢ dist a b = dist b c ∧ dist a b = dist a c
```

To break those two things into their own goals, we use the tactic `constructor`. We now have TWO goals, the first says

```
⊢ dist a b = dist b c
```

and down below is a second goal (with the same assumptions), which says:

```
⊢ dist a b = dist a c
```

To work on just the first goal, we write `·` which hides the goals you're not working on.
So solve the first goal, we need to prove:

```
Step6 : dist b a = dist b c
⊢ dist a b = dist b c
```

If we could somehow convert `Step6` to the goal, we'd be making progress. The syntax for that is `convert`. (Convert might try to do too much, and you can control that with `using`). We wrote:

```
convert Step6 using 1
```

and the goal state became:

```
⊢ dist a b = dist b c
```

At the last step, we can give the exact proof of what it's asking. That's called `exact`.
Here's as far as we got:

```
import Mathlib

class EuclideanPlane where
  Pt : Type
  Line : Type
  Circ : Type
  Pt_on_Line : Pt → Line → Prop
  Line_of_Pts : ∀ a b : Pt, ∃ L : Line, (Pt_on_Line a L) ∧ (Pt_on_Line b L)
  Pt_on_Circ : Pt → Circ → Prop
  Center_of_Circ : Pt → Circ → Prop
  Circ_of_Pts : ∀ a b : Pt, ∃ α : Circ, (Center_of_Circ a α) ∧ (Pt_on_Circ b
α)
  dist : Pt → Pt → ℝ
  dist_symm : ∀ (a b : Pt), dist a b = dist b a
  Circs_inter : Circ → Circ → Prop
  Pts_of_Circs_inter : ∀ α β : Circ, Circs_inter α β → ∃ a b : Pt,
    Pt_on_Circ a α ∧ Pt_on_Circ b α ∧ Pt_on_Circ a β ∧ Pt_on_Circ b β ∧ a ≠
b
  Pt_in_Circ : Pt → Circ → Prop
  Circs_inter_iff : ∀ (α β : Circ), Circs_inter α β ↔
    (∃ (a b : Pt), Pt_on_Circ a α ∧ Pt_in_Circ a β ∧
      Pt_on_Circ b β ∧ Pt_in_Circ b α)
  Pt_on_Circ_iff : ∀ (a b c : Pt) (α : Circ), Center_of_Circ a α →
    Pt_on_Circ b α → (Pt_on_Circ c α ↔ dist a c = dist a b)
  Pt_in_Circ_iff : ∀ (a b c : Pt) (α : Circ), Center_of_Circ a α →
    Pt_on_Circ b α → (Pt_in_Circ c α ↔ dist a c < dist a b)

variable [EuclideanPlane]

namespace EuclideanPlane

def IsEquilateralTriangle (a b c : Pt) : Prop := (dist a b = dist b c) ∧
(dist a b = dist a c)

theorem PropIp1 (a b : Pt) : ∃ (c : Pt), IsEquilateralTriangle a b c := by
  have Step1 : ∃ (α : Circ), Center_of_Circ a α ∧ Pt_on_Circ b α :=
    Circ_of_Pts a b
  obtain ⟨α, a_center_of_α, b_on_α⟩ := Step1
  have Step2 : ∃ (β : Circ), Center_of_Circ b β ∧ Pt_on_Circ a β :=
    Circ_of_Pts b a
```

```
  obtain ⟨β, b_center_of_β, a_on_β⟩ := Step2
  have Step3 : Circs_inter α β := by
    -- I don't want to prove this right now; let's see if we can
    -- complete the proof, postponing this part. That's called `sorry`
    sorry
  have Step4 : ∃ (c d : Pt), Pt_on_Circ c α ∧ Pt_on_Circ d α ∧
    Pt_on_Circ c β ∧ Pt_on_Circ d β ∧ c ≠ d :=
      Pts_of_Circs_inter α β Step3
  obtain ⟨c, d, c_on_α, d_on_α, c_on_β, d_on_β, c_ne_d⟩ := Step4
  -- next step: create a line through `a` and `c`
  --have Step5 : ∃ (L : Line), Pt_on_Line a L ∧ Pt_on_Line c L :=
Line_of_Pts a c
  --obtain ⟨L, a_on_L, c_on_L⟩ := Step5
  have Step5 : dist a c = dist a b :=
    (Pt_on_Circ_iff a b c α a_center_of_α b_on_α).1 c_on_α

  have Step6 : dist b a = dist b c :=
    (Pt_on_Circ_iff b c a β b_center_of_β c_on_β).1 a_on_β

  use c
  unfold IsEquilateralTriangle
  constructor
  · convert Step6 using 1
    exact dist_symm a b
  ·  --- HOMEWORK
    sorry
```